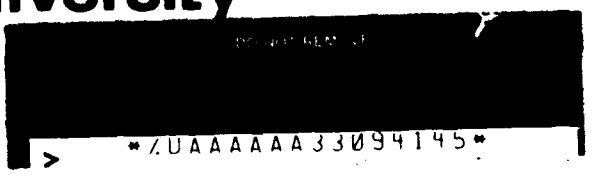
CMU-CS-80-106

An Experimental Analysis of the B* Tree Search Algorithm

Andrew J. Palay
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa. 15213
8 March 1980

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University



An Experimental Analysis of the B* Tree Search Algorithm

Andrew J. Palay
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa. 15213
8 March 1980

Abstract

We present a set of selection rules for guiding the B* search procedure. The selection rules are based on a simple probability model. Using those selection procedures, the B* algorithm will expand approximately one-third as many nodes as a best first search and approximately two-thirds as many nodes as the previous best version of the B* algorithm. The B* algorithm, using the selection procedures, will expand 30% more nodes than optimum, for small trees, and up to 170% for larger trees. We also examine the issues of when the B* paradigm is useful and the effect of invalid bounds on the algorithm.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

1. Introduction

The B* tree search algorithm [Berliner 79] is a *proof* procedure for determining the best move at the root of the search tree. Associated with each node in the tree is an optimistic value and a pessimistic value. Those values bound the *true* value of the node. The object of the B* algorithm is to prove that the true value of one of the nodes adjacent to the root is greater than or equal to the true values of the remainder of the nodes adjacent to the root. This is accomplished by showing that the pessimistic value of one of the nodes adjacent to the root is greater than or equal to the optimistic values of the rest of the nodes adjacent to the root. The move to that node is then guaranteed to be the best.

Several open questions exist with respect to the B* tree search algorithm. The first question deals with the selection of nodes to explore. Unlike other search algorithms, the B* algorithm allows for a great deal of freedom in the selection of nodes to explore. Berliner [Berliner 79] showed that by exploring nodes, other than the best, at the top level of the search tree, the size of the search tree can be reduced to about 60% of the size of the comparable best-first search. In this paper we will present a set of rules to guide the search at all levels of the tree. A new set of rules is presented to choose which node to explore at the top level of the tree, as well as an additional set of rules for selecting nodes for exploration at the lower levels of the search tree. It will be shown that the best node at each level of the search tree is not necessarily the proper node to explore. In fact, it is possible that the best node will never be explored. This is unlike other search procedures, which ultimately explore the best node.

Another question to be answered is when is the B* algorithm useful? For example, a best-first search procedure is equally as efficient in both adversary and non-adversary searches. However, the efficiency of a depth-first search depends on whether or not there is an opponent. With an opponent, alpha-beta pruning can be used to increase the efficiency of the search. Without an opponent, no pruning can be done. We will examine the effect that having or not having an opponent has on the B* algorithm, as well as the general conditions that must be satisfied in order for the B* algorithm to be useful.

For the B* algorithm to be a *proof* procedure it is necessary for the optimistic and pessimistic values to bound the true value of the node. In particular, the evaluation function used by the B* algorithm must return values that bound the true value of the node. The final question concerns the effect on the results of the B* algorithm, when the true value of a node is found outside the bounds retuned by the evaluation function.

2. The B* Algorithm - An Example

We begin with a presentation of the B* algorithm. The algorithm used in this paper is a modified version of the original algorithm presented by Berliner. The algorithm has two sections: one which pertains to decisions made at the top level of the search tree, and a second which pertains to decisions made at lower levels in the tree.

The example presented is of a two person (adversary) search. We shall refer to the side on move at the root as the "player" and the other side as the "opponent".

The B* algorithm begins by expanding the root of the search tree. Figure 2-1 shows the initial state of the example. Each node has two values. The first value is the optimistic value of the node and the second is the pessimistic value of the node. Those values bound the true value of the node. The optimistic value is the highest value the true value can assume. The pessimistic value is the lowest value that the true value can assume. Given that the true value is bounded by the optimistic and pessimistic value, a node adjacent to the root is guaranteed to be the best whenever the pessimistic value of that node is greater than or equal to the optimistic value of all its sibling nodes. The search is terminated at that point.

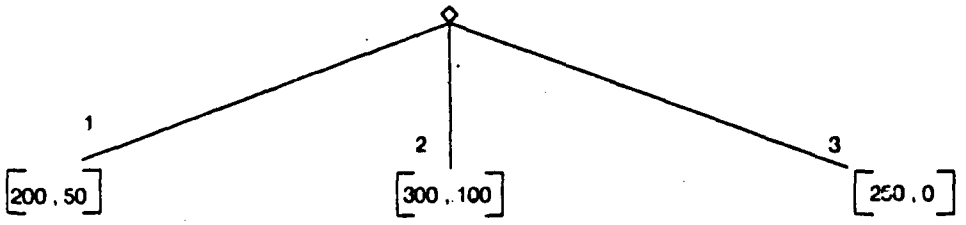


Figure 2-1: Beginning of Search

The termination condition can be specified another way. Given the descendants, $N_{i,1}$, ... , $N_{i,t}$, of a node N_i, with ranges $[\alpha_{i,j}, \beta_{i,j}]$, a node $N_{i,j}$ is *relevant* to N_i, if for all nodes $N_{i,k}$, $k \neq j$, $\alpha_{i,j} > \beta_{i,k}$. If $\alpha_{i,j} \leq \beta_{i,k}$ for some node $N_{i,k}$, then the true value of node $N_{i,j}$ is guaranteed to be less than or equal to the true value of node $N_{i,k}$. Node $N_{i,j}$ can have no further effect on the search and can be disregarded. The search can be terminated whenever there is only one relevant node to the root of the tree.

The B* algorithm produces a *separation point* between the best node and the remaining nodes. The true value of the best node is guaranteed to be greater than or equal to the separation point, and the true values of the remaining nodes are be guaranteed to be less than or equal to the separation point.

Every time the search returns to the top level of the tree, the termination condition is checked. Whenever the termination condition is not satisfied the top level search procedure must decide which node to explore and under which strategy. There are two search strategies available. The search

procedure can either:

1. Attempt to raise the pessimistic value of the best node above the optimistic values of the remaining nodes. This strategy shall be refered to as the PROVEBEST strategy after [Berliner 79].

2. Attempt to lower the optimistic value of the remaining nodes to below the current pessimistic value of the best node. This strategy shall be refered to as the DISPROVEREST strategy also after [Berliner 79].

In the example of figure 2-2 the termination condition is not met. We begin by trying to show that node 2 is the best node using the PROVEBEST strategy. Under the PROVEBEST strategy only the best node adjacent to the root may be explored (with exception of nodes with equal optimistic values).

The lower level procedure takes over at this point. The role of this procedure is to guide the search through the existing search tree until a leaf node is encountered. It then expands that node and recomputes that node's optimistic and pessimistic values. Whenever either bound of a node changed, the search is backed up to its parent. If the search has been backed up to the root of the search tree then top level procedure resumes control, otherwise the lower level procedure continues control. In the example the current node is a leaf node and therefore it is expanded. Figure 2-2 presents the status of the tree following the expansion, before any values are backed up. Notice that the sign of the values of the new node are reversed. Again the optimistic value is still listed first. As a result of the expansion, the bounds at node 2 are now changed. The best that the opponent can do is -100 (the maximum of the optimistic values). His best value defines the player's worst value (reversing the sign). Thus the pessimistic value of node 2 remains at 100. However, the worst the opponent can do (the maximum of the pessimistic values) is -190, by choosing node 5. Thus, the best the player can do at node 2 is 190. The state after backing up these values is given in figure 2-3.

The example search is now back at the top level. The termination condition is not met. All the nodes are still relevant to the root. Node 3 is now the current candidate for being the best node. In order to show that node 3 is the best node, the pessimistic value of node 3 must be raised above 200. Using the PROVEBEST strategy, node 3 is expanded, yielding the tree in figure 2-4.

This time the current bounds of node 3 reflect the bounds given by its descendants. Thus no backup is done and the lower level procedure must decide which of the nodes 7, 8, and 9 should be explored. Since the current goal of the search is to raise the pessimistic value of node 3 to above 200, it would be unnecessary to explore node 9. In order to raise the pessimistic value of node 3 to above 200 it is only necessary to lower the optimistic value of nodes 7 and 8 below -200. The lower level procedure is responsible for making that selection. Assuming that node 7 is selected, the tree after expansion is given in figure 2-5. Given the opponent's choice of node 7, the best the player can do is a value of 250 (choosing node 10) and the worst he can do is a value of 200 (choosing node 11). Thus

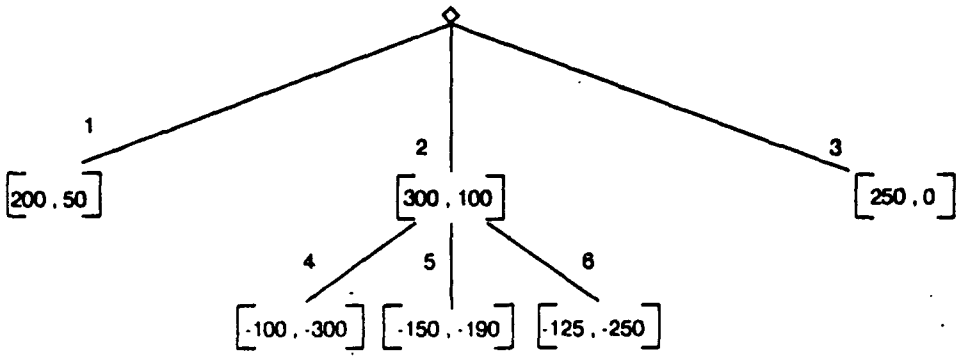


Figure 2-2: PROVEBEST Strategy - Expansion of Node 2

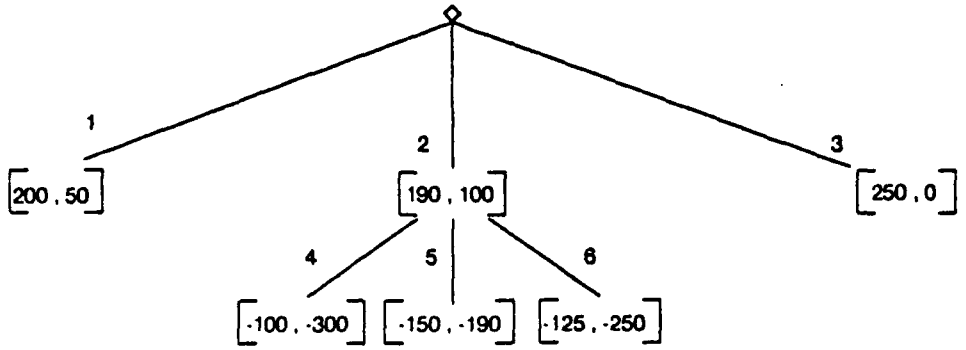


Figure 2-3: PROVEBEST Strategy - Backup of Values

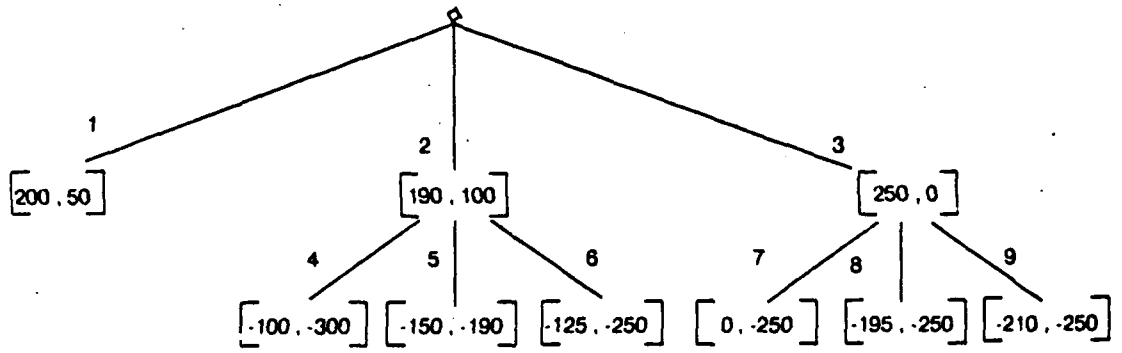


Figure 2-4: PROVEBEST Strategy - Expansion of Node 3

the best the opponent can do is -200 and the worst he can do is -250. The best he can do given the selection of node 3 is -195 (choosing node 8) and the worst is still -250. The backed up bounds at node 3 are 250 and 195. This is reflected in figure 2-6.

The search is again back at the top level. This time the DISPROVEREST strategy is employed. The only node that is still in contention with node 3, is node 1. Node 2 is no longer relevant to the root and therefore can be ignored. Node 1 is expanded by the lower level procedure. Backing up the values from the expansion yields a new bound of [80,0] for node 1 (figure 2-7). Again control returns to the

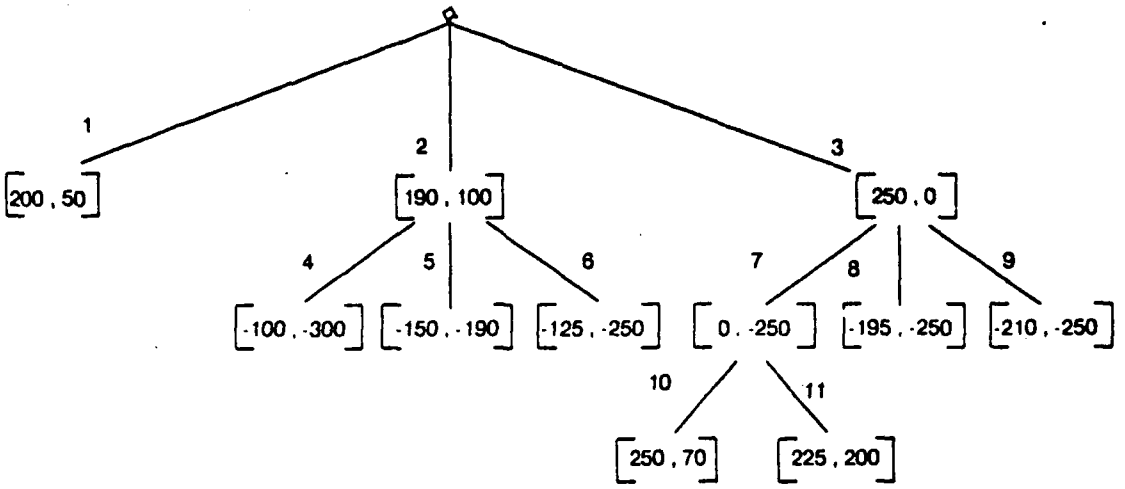


Figure 2-5: PROVEBEST Strategy - Expansion of Node 7

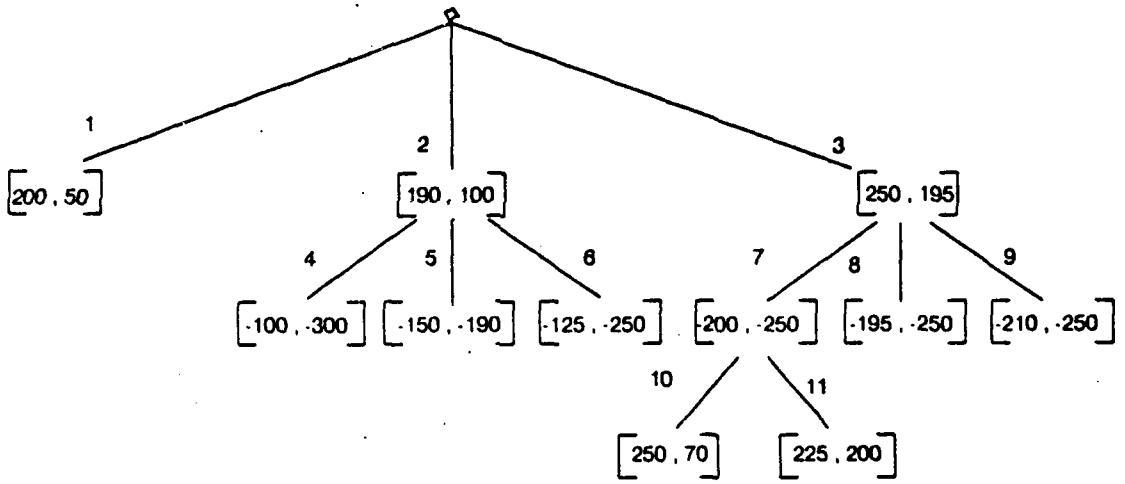


Figure 2-6: PROVEBEST Stagegy - Backup of Values to Node 3

top level procedure and this time the termination criterion is met. The search is completed with node 3 returned as the best node.

The B* search algorithm is given in figure 2-8. The algorithm for a non-adversary search is almost identical. There is no alternation of sign, and no alternation of optimistic and pessimistic values.

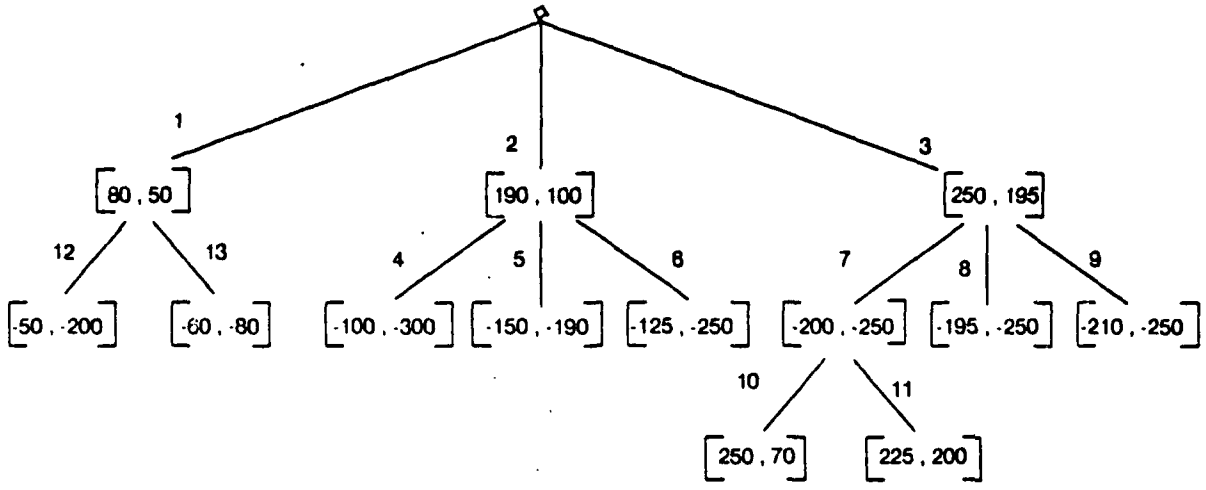


Figure 2-7: Final Tree - After DISPROVEREST Strategy on Node 1

```
PROCEDURE BSTAR!TOP!LEVEL(starting!position);
BEGIN

  expand top level of tree from starting!position;

  WHILE pessimistic value of best node < highest optimistic
        value of alternative nodes DO
    BEGIN
      select search strategy, either PROVEBEST or DISPROVEREST;
      current!node ← next node to explore;
      BSTAR!LOWER!LEVEL(current!node);
    END;

  RETURN(best node);

END;


PROCEDURE BSTAR!LOWER!LEVEL(current!node);
BEGIN
  IF current!node has not been expanded THEN
    expand tree from current!node;

  new!optimistic!value ← -(maximum pessimistic value
                           of descendants of current!node)
  new!pessimistic!value ← -(maximum optimistic value
                            of descendants of current!node);

  WHILE new!optimistic!value=current optimistic value of current!node AND
        new!pessimistic!value=current!pessimistic!value of current!node DO
    BEGIN

      next!node ← next node to explore from descendants of current!node;
      BSTAR!LOWER!LEVEL(next!node);
      new!optimistic!value ← -(maximum pessimistic value
                               of descendants of current!node);
      new!pessimistic!value ← -(maximum optimistic value
                                of descendants of current!node);
    END;

  current!optimistic!value of current!node ← new!optimistic!value;
  current!pessimistic!value of current!node ← new!pessimistic!value;

END;
```

Figure 2-8: The B* Algorithm for Adversary Searches

3. Decisions at the Top Level of the Tree

Two decisions are made at the top level of the search. The first decision is the choice of the search strategy. Then, once the strategy is chosen, a decision on which node to explore is made. The selection of the search strategy is the most important of the two decisions. The ability to use both strategies is the key to the effectiveness of the B* algorithm. Whereas a best-first search must pursue the best sub-tree (PROVEBEST), B* can, at the proper moment, use the DISPROVEREST strategy to advantage to produce the necessary separation.

The procedure for choosing the search strategy can be viewed from two levels. From a global viewpoint, the strategy selection procedure can be measured on its ability to split the effort between strategies. In most cases, it is important to use both strategies. It is rare that one strategy is used to the exclusion of the other. From a local viewpoint, the strategy selection procedure can be measured on how well it chooses the better strategy every time the search returns to the top level. Even though, from a global viewpoint, the use of strategies should be mixed, from a local viewpoint there are clear choices as to which strategy should be used. A proper strategy selection procedure should be effective from both viewpoints.

3.1 Trees With Two Relevant Nodes

Let us begin by examining the strategy selection, given that there are only two relevant nodes. Later the derived procedure will be expanded to situations where there are more than two relevant nodes at the top level. In our simple case, there are two relevant nodes, N_1 and N_2. The ranges of the two nodes are $[\alpha_1,\beta_2]$ and $[\alpha_2,\beta_2]$ respectively. Also, assume that $\alpha_1 \geq \alpha_2$ and that $\alpha_2 > \beta_1$. There are a total of six cases that can occur when choosing the search strategy. These are:

1. $\alpha_1 = \alpha_2$ and $\beta_1 > \beta_2$
2. $\alpha_1 = \alpha_2$ and $\beta_1 = \beta_2$
3. $\alpha_1 = \alpha_2$ and $\beta_1 < \beta_2$
4. $\alpha_1 > \alpha_2$ and $\beta_1 > \beta_2$
5. $\alpha_1 > \alpha_2$ and $\beta_1 = \beta_2$
6. $\alpha_1 > \alpha_2$ and $\beta_1 < \beta_2$

For each of these cases we shall develop a rule for determining which strategy is to be selected. The choice of strategies will be based on two basic principles:

1. If an action needs to be taken sometime in the search then do it immediately. If that action were postponed, unnecessary work may be performed. By taking the action immediately, certain information may be obtained that alters the current assumptions of the search, making exploration of other nodes unnecessary.

2. Attempt to choose the strategy that has the highest probability of total success. This is the key to the selection rules. Choices are made to maximize the probability of success, *without regard to the amount of work that needs to be done.* The notion of total success is important. It is possible to make selections based on partial success. In selecting strategies, one could make the choice based on the probability of moving a value part way to the goal. It would then be possible to choose a strategy, move the value part way to the goal and then be required to use the other strategy, which originally had the highest probability of total success. Use of the other strategy could result in reaching the original goal. The work done under the first strategy would have been wasted.

In addition, one assumption is made about the true value of a node in relation to its bounds. It is assumed that the true value of a node is uniformly distributed over the range of that node. This assumption, to be referred to as the *uniform assumption*, is made throughout the entire search, not only at the top level.

In case 1, it maybe possible to prove that both nodes are the best. However given the "uniform assumption" it is more likely that the best node will be N_1. The choice is either to use the PROVEBEST strategy on node N_1 or use the DISPROVEREST strategy on node N_2. Using principle 2, the strategy chosen should be the DISPROVEREST strategy. By using the *uniform assumption* the probability of total success of using the PROVEBEST assumption is lower than the probability of total success using the DISPROVEREST strategy because the only way the search can succeed using the PROVEBEST strategy is by raising the pessimistic value of the best node up to its optimistic value. Case 3 is symetric to case 1.

In case 2, both nodes are identical. It is possible to prove that either node is the best, using either strategy. Unfortunately there is no reasonable way to choose which node to explore or which strategy to use. What is needed is a third strategy, that of EXPLORE. This strategy would attempt to provide the most information from the lower levels of the tree without regard to the goal of proving one node better than the other. In reality using either strategy will provide just that information.

Case 4 is the most difficult case for properly deciding the search strategy. In this case the two ranges overlap with a definite best node. It is possible, using either strategy, to show that node N_1 is the best node. The *uniform assumption* is the key to making the correct decision in this case. By principle 2, the strategy that maximizes the probability of success should be selected. That strategy will also minimize the probability of failure. Given the *uniform assumption* the strategy selection becomes very simple. For the best node calculate the probability that the true value is less than or equal to the optimistic value of node N_2. This is the probability of failure of using the PROVEBEST

strategy. That probability is given by:

$$P_{Fail\text{-}Prove} = (\alpha_2 - \beta_1) / (\alpha_1 - \beta_1) \quad (3.1)$$

Then for node N_2 calculate the probability of failure for using the DISPROVEREST strategy. This is the probability that the true value of node N_2 is greater than or equal to the pessimistic value of the best node. That probability is given by:

$$P_{Fail\text{-}Disprove} = (\alpha_2 - \beta_1) / (\alpha_2 - \beta_2) \quad (3.2)$$

Now if $P_{Fail\text{-}Prove}$ is less than $P_{Fail\text{-}Disprove}$ the PROVEBEST strategy should be chosen. Similarily, if $P_{Fail\text{-}Disprove}$ is less than $P_{Fail\text{-}Prove}$ then the DISPROVEREST strategy should be chosen. If the values are equal (not very likely) either strategy can be selected. In that case it might be reasonable to select the strategy least recently used; thereby splitting the effort between the two strategies.

Case 5 is symmetric to case 1. In this case either case can be disproved. However, it is possible to prove node N_1 to be the best using the PROVEBEST strategy. In this case, the PROVEBEST strategy will always have a higher probability of success than the DISPROVEREST strategy.

In case 6 node the range of node 1 completely surrounds the range of node N_2. In this case the probability of total success of the DISPROVEREST strategy is 0. It is not possible to lower the optimistic value of N_2 below the pessimistic value of N_1. Thus the only strategy available is the PROVEBEST strategy. Even if one relaxes the constraint of principle 2, it would still be necessary to choose the PROVEBEST strategy. No matter how the strategy is selected in other cases the choice of the PROVEBEST strategy in this case would still be mandated by principle 1. Sooner or later one would have to explore node N_1 so it should be done immediately.

3.2 Trees With More Than Two Relevant Nodes

Now let us extend the above rules to the case where there are more than two nodes relevant to the root. Let nodes $N_1, ..., N_t$ be the set of relevant nodes to the root. The range of N_i is $[\alpha_i, \beta_i]$. Assume that the nodes are ordered by optimistic value, such that $\alpha_i \geq \alpha_{i+1}$.

In the case where there are two or more nodes with the highest optimistic value ($\alpha_1 = \alpha_2 = ... = \alpha_k$, for some $k \geq 2$), the DISPROVEREST strategy should be chosen. The node that should be explored is the node with the lowest pessimistic value (of the nodes with the same optimistic values). Attempting to disprove any other node with lower optimistic values would still leave the search in a position where the tied nodes would need to be explored. Thus by principle 1 those nodes should be explored first.

In the case where the range of a node is embedded within the range of the best node (for some k,

$\beta_1 \leq \beta_k$), the PROVEBEST strategy should be selected. A problem arises here in that this case and the above case are not mutually exclusive. It is possible that two nodes have the highest optimistic value and a third node is totally embedded in both of the nodes. In this case one rule specifies that the DISPROVEREST strategy should be chosen, where as the other rule selects the PROVEBEST strategy. Either strategy could be used, with about equal success. To maintain consistency in the selection of which node to expand (see below), the DISPROVEREST strategy should be chosen.

The last possibility is that there exists only one node with the highest optimistic value ($\alpha_1 > \alpha_2$) and all the remaining nodes have pessimistic values less than the pessimistic value of the best node ($\beta_1 > \beta_k$, k=2,3,...,t). This is the extension of case 4. For the best node calculate the value given in equation (3.3).

$$P_{Fail\text{-}Prove} = (\alpha_2 - \beta_1) / (\alpha_1 - \beta_1) \tag{3.3}$$

This gives the probability of failure in using the PROVEBEST strategy on node 1. For each of the nodes $N_2, \ldots, N_t$, calculate the value given in equation (3.4).

$$F_i = (\alpha_i - \beta_1) / (\alpha_i - \beta_i) \tag{3.4}$$

Then add those values together, yielding $P_{Fail\text{-}Disprove}$. If the $P_{Fail\text{-}Prove}$ is less than $P_{Fail\text{-}Disprove}$, then choose the PROVEBEST strategy, otherwise choose the DISPROVEREST strategy.

The value $P_{Fail\text{-}Disprove}$ does not really reflect a probability measure of the failure of the strategy[1]. However, it does reflect the fact that given more than one node to disprove, additional work needs to be done by using the DISPROVEREST strategy than by using the PROVEBEST strategy. Thus, unless the DISPROVEREST strategy has a high probability of success it is better to choose the PROVEBEST strategy until the number of remaining nodes is small.

3.3 Strategy Selection From a Global View

This completes the examination the choice of strategies from the local viewpoint. From the global view the set of selection rules provides for a reasonable mixture of the use of the two strategies. Assume there are two nodes, N_1 and N_2, with ranges [200,100] and [150,0]. By case 4, the PROVEBEST strategy would be used on node N_1. Say that upon returning from exploring N_1, the new range of N_1 becomes [180,130]. This time the DISPROVEREST strategy should be used on node N_2. Say the result of that selection is a range for N_2 of [135,100]. Now the PROVEBEST strategy is selected. The effect of the selection rules is to choose dynamically a separation point between the nodes and move the pessimistic value of the best node above that value and the optimistic values of the rest of the nodes below that value. In the above example the separation point is around 135.

---

[1] The probability of failure of the DISPROVEREST strategy is equal to $(1-(\prod_{i=2\ldots t}(1-F_i)))$.

Several attempts were made to estimate, a priori, the separation point of the search. Although reasonable estimates could be made, any search using an estimate failed. When an a priori separation point was used, the search would ultimately get locked into one choice of strategy. No reasonable criterion on when to choose a new separation point could be determined. The above approach of dynamically choosing the separation point, will guide the search towards a reasonable separation point, but will not get locked into one strategy or the other.

3.4 Choice of Nodes to Explore

Finally, there is the question of which node to explore under each strategy. The node to be explored under the PROVEBEST strategy is the node with the highest optimistic value. If two or more nodes have the highest optimistic value the PROVEBEST strategy will not be used. Under the DISPROVEREST strategy the node with the second highest optimistic value should be explored. For the search to be completed, either the optimistic value of the second best node would have to be lowered below the pessimistic value of the best node, or the pessimistic value of the best node would have to be raised above the optimistic value of the second best node. If any other node was explored the above action would still have to be done. Thus, given the choice of the DISPROVEREST strategy, by principle 1, the node with the second highest optimistic value should be explored.

4. Decisions at the Lower Level of the Tree

The lower level procedure must choose which descendant of the current node to explore. That choice is made under four conditions. The search may be proceeding under either the PROVEBEST strategy or the DISPROVEREST strategy. Then for each strategy, it may be the player's move or the opponent's move. We shall begin by discussing the PROVEBEST case.

Under the PROVEBEST strategy the goal is to raise the pessimistic value of the top level node above the highest optimistic value of the alternatives (we shall refer to that value as Λ). Thus, whenever it is the player's choice, it is necessary and sufficient to raise the pessimistic value of only one of the nodes above Λ. When it is the opponent's choice, it is necessary to lower the optimistic value of all the nodes below $-\Lambda$.

One possible method for selecting the next node to explore is to choose the node with the highest optimistic value. This was the lower level selection procedure originally used by Berliner [Berliner 79]. This approach reflects a view of searching that is not maintained in the B* search paradigm. Under most other search techniques, the search attempts to find the best line of play for both sides. This is not the case for the B* algorithm. For example, when it is the player's choice, the node whose pessimistic value is shown to be greater than Λ may or may not be the best move at that point.

For the player's choice of nodes, the lower level selection procedure should explore the node with the greatest probability of success. Using the *uniform assumption*, the probability of successfully raising the pessimistic value for any of the nodes to at least Λ is given in equation (4.1), assuming the node has a range of $[\alpha_i,\beta_i]$.

$$P_i = (\alpha_i - \Lambda) / (\alpha_i - \beta_i) \qquad (4.1)$$

The node with the highest P_i, should be chosen.

For the opponent's choice of nodes, the selection should be based on the probability of failure. Since the goal is to lower the optimistic value of all the opponent's nodes below $-\Lambda$, the best node to choose is the one which has the highest chance of failing. This selection strategy attempts to minimize the amount of unnecessary work.

If the pessimistic value of one of the opponent's nodes is raised above $-\Lambda$, the optimistic value of its parent node will drop below Λ. Thus, it will not be possible to complete the search by exploring that node. A different node, for the player, will have to be explored. If all the nodes, for the player, can be shown to have optimistic values less than Λ, then the pessimistic value of their parent node rises above $-\Lambda$. This can continue to the top level node. At the top level, the node just searched will no longer be the best node, and the search will be re-directed by a new application of the top level decision rules.. By choosing the node with the highest probability of failure that process is hastened.

Note, at no time is work being wasted. The nodes being explored would have to be explored sooner or later in the search, if the current top level node is indeed the best node.

One measure for the probability of failure is:

$$P_i = (-\Lambda - \beta_i) / (\alpha_i - \beta_i) \quad (4.2)$$

This measure unfortunately takes a very shortsighted view of the search. Using it, the search often degrades into a race of lowering optimistic values. For example,say at the top level there are two nodes N_1 and N_2, with ranges [200,50] and [199,40]. Using the above measure a search of N_1 using the PROVEBEST strategy could yield a range where the new optimistic value is just under the optimistic value of N_2, say [195,50]. Then exploring N_2 could yield a new optimistic value just under the optimistic value of N_1. Then the process returns again to N_1. When this occurs, the search expands a large number of nodes without achieving a great deal of progress.

In order to solve this problem it is necessary to force the selection procedure to take a more far sighted view. Instead of trying to pick the node with a good chance at raising the pessimistic value by a little, the selection procedure can attempt to choose a node with a good chance of raising the pessimistic value a great deal. At the top level, by moving the optimistic value of the old best node just below the optimistic value of the new best node, it is still quit likely that the old best node will still be the true best node. However, if the optimistic value could be lowered to a position closer to the pessimistic value of the other node, it will be much more likely that the old best node will not be the true best node. Thus instead of using Λ in equation (4.2), it might be more reasonable to use the maximum of the pessimistic value of the current best node and the highest pessimistic value of the alternative (call this value Γ). The search would then be either attempting to disprove the current best node completely or at least attempting to reduce it to its lowest point value.

Unfortunately that measure suffers by trying to attain too much. Given a set of nodes for the opponent to choose from, only the node with the highest optimistic value would have a chance of moving the pessimistic value up to Γ.

In the simulations it has been shown that a reasonable selection procedure for the opponent's level is to select the node with the highest probability of raising the pessimistic value above the midpoint of Λ and Γ. The new measure of failure is:

$$P_i = ((\Lambda + \Gamma)/2 - \beta_i) / (\alpha_i - \beta_i) \quad (4.3)$$

That value allows the search to push for major gains, while still allowing nodes without the highest optimistic value to be explored early.

We now turn to the decision rules for the DISPROVEREST strategy. When it is the player's choice of moves, it is necessary to lower the optimistic values of all the nodes below the pessimistic value of the best top level node (Λ). For the opponent's choice, it is necessary and sufficient to raise the pessimistic value above $-\Gamma$. This is symmetric to the rules under the PROVEBEST strategy. By the similar arguments as above, the node to choose for the player, is node that maximizes:

$$P_i = (\alpha_i - (\Lambda + \Gamma)/2) / (\alpha_i - \beta_i) \quad (4.4)$$

and the node to choose for the opponent, maximizes:

$$P_i = (-\Gamma - \beta_i) / (\alpha_i - \beta_i). \quad (4.5)$$

5. Simulation Results

Several versions of the B* algorithm were simulated in order to compare the effectiveness of each version. Five versions of the search algorithm were tested:

1. *Best-First* (*BF*): This was the base run. The algorithm used only the PROVEBEST strategy and always selected the best node throughout the search.

2. *Probability Based - Lower Levels Only* (*LL*): This version also only used the PROVEBEST strategy but used the selection rules given in section 4 at lower levels of the tree to guide the search.

3. *Depth Based - Top Level Only* (*DB*): This is the method for selecting the search strategy presented by Berliner [Berliner 79]. If the sum of the squares of the depth from which the optimistic bounds of the alternative nodes had been backed up is less than the square of the depth the pessimistic value of the best node had been backed up then the best alternative is chosen, otherwise the best node was explored. At the lower levels of the tree the best node was always explored.

4. *Probability Based - Top Level Only* (*TL*): This version used the method for selecting strategies presented in section 3. At the lower levels of the tree, the best node was explored.

5. *Probability Based - All Levels* (*AL*): This uses the probability based selection procedures at all levels of the tree.

The simulation procedure used the algorithm for generating canonical trees presented by Berliner [Berliner 79], with one major modification. Whenever the range of a newly expanded node was of size 2 or less, the range was reduced to a point value. This greatly reduced the maximum depth of the best-first trees. If the range of a node is small, the probability that the range will be narrowed by the next expansion is quite small. Thus ranges of 2 or less may be pursued a very long time with no useful results. The maximum depth of any best-first search in the simulations was 38. Without collapsing the ranges, a greater number of large best-first trees would have been generated[2]. Those trees would normally be solved in significantly fewer expanded nodes by any version that used both search strategies.

Each simulation run consisted of searches of 3200 trees. The possible ranges for the top level nodes were selected from values between 0 and four different values (200, 800, 3200 and 12800). For each of the ranges the branching factor was varied from 3 to 8 by increments of 1. For each (range, branching factor) pair, 100 trees were searched.

---

[2] In Berliner's simulations where no collapsing of ranges took place, the maximum depth of a best-first tree was 98 and there were 226 intractable trees out of a sample of 1600 trees. Trees were labeled intractable if the search proceed past a depth of 100 or if more than 30000 nodes were created.

Three measures were calculated for each search:

1. the number of nodes expanded.

2. the number of nodes explored.

3. the maximum depth of the search.

The number of nodes expanded is the most important measure. In most cases the major cost of the search will be in the expansion of the nodes and the calculation of the evaluation function. However, the number of nodes explored can be important. The number of nodes explored counts the number of times a decision of which node to next explore is made. This is effectively a count of the amount of downward movement that is made in the search. The higher the value (given the number of expanded nodes remains constant) the more often the search is making small changes in values. The number of explored nodes would be important in an application where the current state of each node could not be maintained. Thus, whenever a downward movement is made the move generator would have to be run to generate the next set of successors.

For each of these measurements a comparison was made to the results of the best-first version of the algorithm. For the first two measures, the value calculated for each search of the simulation, is the ratio of the value of the test version and the value of the best-first version. The numbers reported are averages over various ranges of the number of nodes expanded in the best-first search. For the depth measurement, the numbers reported are the average maximum depth of the searches, again averaged over the same ranges of the best-first trees[3].

Figures 5-1, 5-2 and 5-3 present the results of the simulations. In figure 5-1, the number of nodes expanded by the *AL* version is significantly fewer than the number of nodes expanded by any of the other strategies. Even on small trees of size less than 50 expanded nodes, the *AL* version does 20% better than the best-first version, where as the *DB* version can only match the effort used by the best-first version. For larger trees the *AL* version does about 35% of the work of the best-first search. The *AL* version also does approximately 35% better than *DB* version. When the selection rules at the lower levels are removed (the *TL* version) the probability based scheme still does better than the *DB* version.

The savings contributed by the lower level selection procedure can also be calculated. On larger trees the *LL* version explores about 25% fewer nodes than the best-first version. The same 25%

[3]The calculation of these values differs from the calculation of Berliner. His figures were calculated by taking the ratio of the sum of the number of nodes expanded in the test version and the sum of the nodes expanded by the best-first version. That calculation allowed the larger trees to have a greater impact on the final figure, than the smaller trees. The above measure attempts to normalize the measurement, prior to averaging

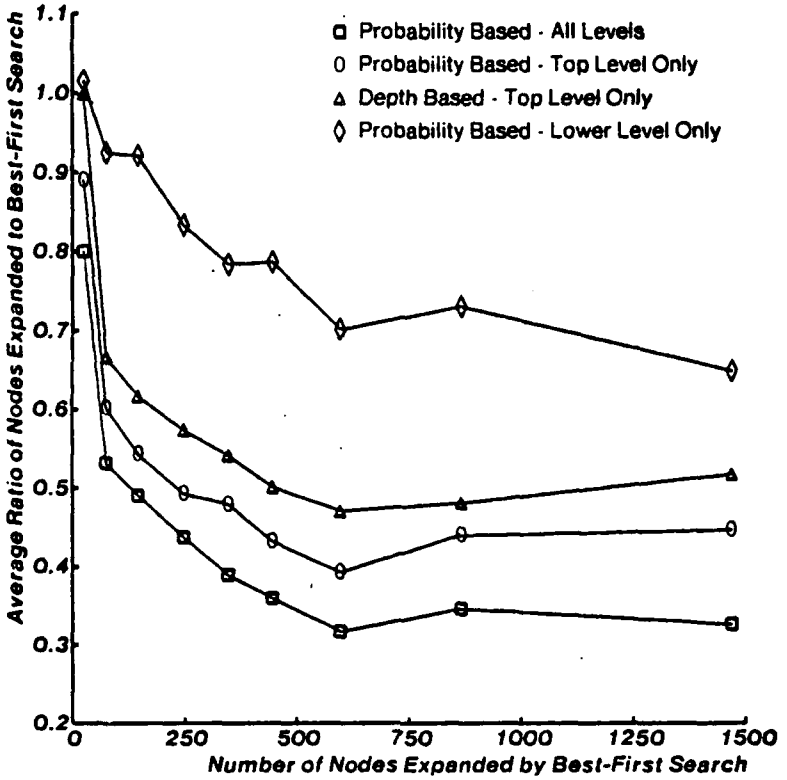


Figure 5-1: Comparison of Nodes Expanded to Best-First

savings can been seen in the comparison of the *TL* version and the *AL* version.

Figure 5-2 gives the number of nodes explored during the search. Again the *AL* version does significantly better than the other versions. Also the *TL* version is strictly better than the *DB* version. The difference between the *TL* version and the *DB* version can be attributed to the difference in the number of nodes expanded. However, the effect of the lower level selection procedures is greater than can be accounted for the number of expanded nodes. For nodes explored the savings of the lower level procedure is about 35%. One of the results of the lower level procedure is to select nodes which will have the greatest impact on the search further up in the tree.

The results of the average depth of the search are presented in figure 5-3. The interesting result of this measurement is that all versions that select both strategies explore the tree to approximately the same depth. Even though the depth based version attempts to "minimize" the depth of the search, the probability based versions do not do significantly worse. Thus, the probability based versions will still reduce the maximum depth of the search, as it attempts to make a reasonable choice for the search strategy.

The various versions of the search algorithm were also compared to the optimum search trees. An algorithm for generating optimum B* search trees is given in appendix I. An optimal tree is defined as

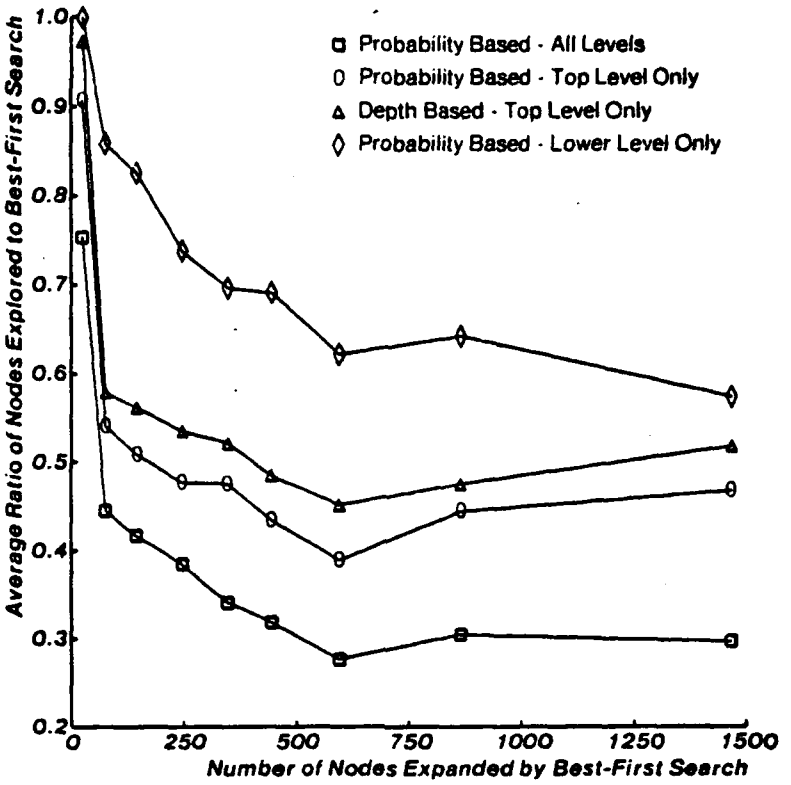


Figure 5-2: Comparison of Nodes Explored to Best First

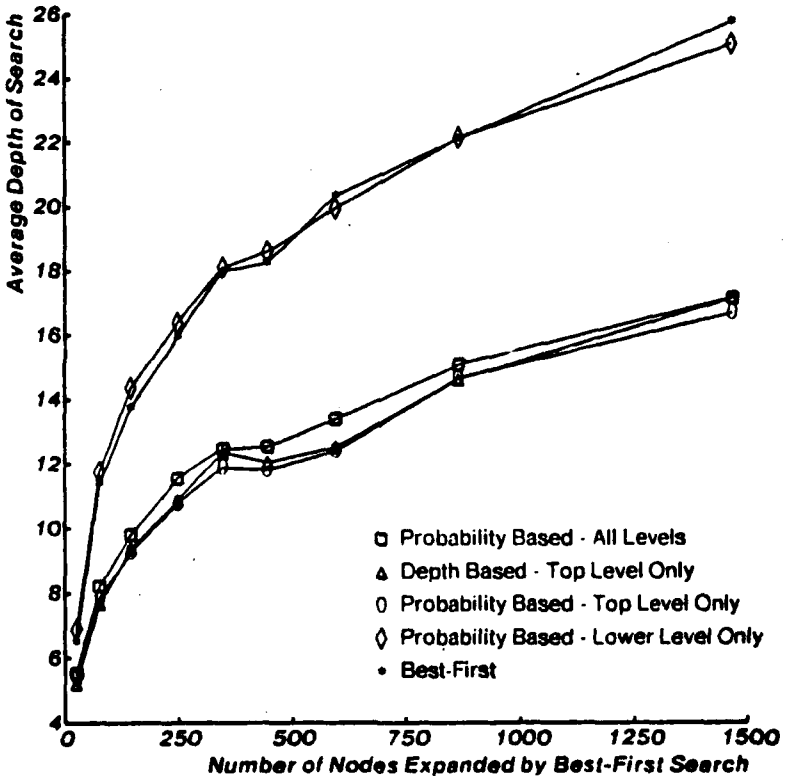


Figure 5-3: Average Depth - Grouped by Best First Search

a tree with the minimum number of expanded nodes that proves which is the best node. It is not possible to calculate the number of nodes explored for optimum trees, as there is no way to determine the proper order of the exploration. Comparisons on the number of nodes expanded and the maximum depth of the search are presented in figures 5-4 and 5-5.

It should be noted that the optimum trees used in the comparison represent a distorted view of the set of trees making up the simulation. In the results discussed below, only the trees for which optimum trees could be calculated, are included. Optimum trees were generated for 2936 out of the 3200 trees simulated. The largest optimum tree calculated expanded 129 nodes. The sample of optimum trees is highly biased towards trees containing fewer than than 50 expanded nodes.

Figure 5-4 presents the comparison with respect to expanded nodes of the *BF*, *DB*, and *AL* versions to the optimum tree. For small optimum trees (10 expanded nodes or less), the *AL* version expands 32% more nodes, compared to 60% more nodes by the *DB* version and 477% for the *BF* version. For larger trees, the *AL* version expands approximately 1 1/2 times more nodes than optimum. For those same trees, the *DB* version expands approximately 2 1/2 times more nodes.

The figures reported for the *BF* version show the effects of the biased sample of optimum trees generated. Of the optimum trees that could not be generated, a majority of the best first searches resulted in the expansion of over 500 nodes. Thus, most of the worst *BF* results were eliminated from the comparison. This resulted in the lower values for the *BF* version for optimum trees of greater than 50 expanded nodes. In comparison, the values eliminated for the *AL* and *DB* versions were more uniformly distributed over the entire spectrum of the number of nodes expanded.

Figure 5-5 compares the *AL* and *BF* versions to the optimum with respect to the depth of the search. For large optimum trees, the *AL* version searches approximately 3 ply deeper than the optimum search.

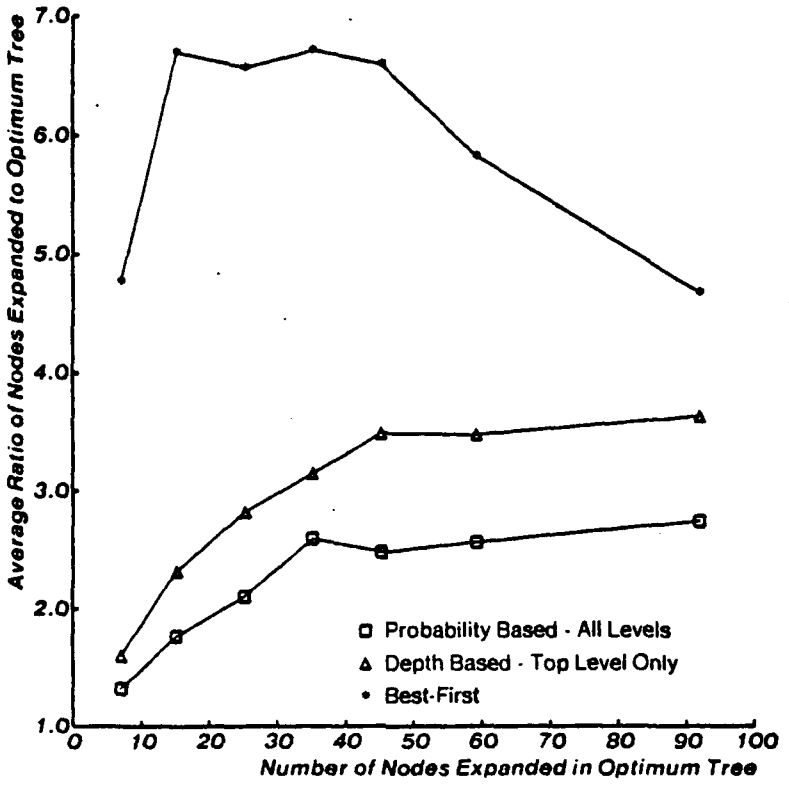


Figure 5-4: Comparison of Nodes Expanded to Optimum

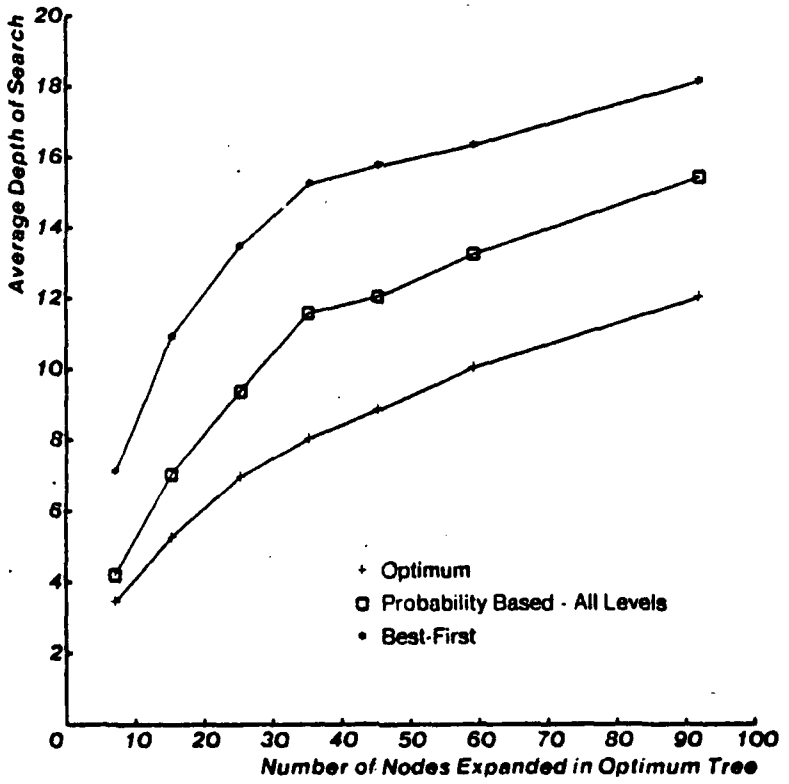


Figure 5-5: Average Depth - Grouped by Optimum Tree Size

6. Adversary vs. Non-Adversary Searches

In the previous sections of this paper we have only considered the use of B* for adversary searches. Berliner also proposed the use the of B* for non-adversary searches. The major advantage of B* over other search procedures is the ability to put some effort into eliminating moves at the top level by lowering the node's optimistic value using the DISPROVEREST strategy. In a non-adversary search the use of the DISPROVEREST strategy is costly and probably non-productive.

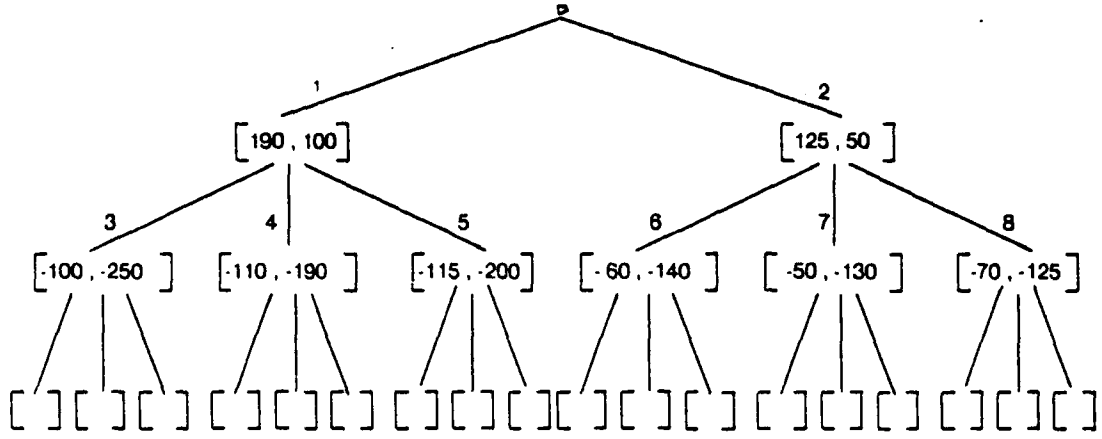


Figure 6-1: Adversary Search Tree

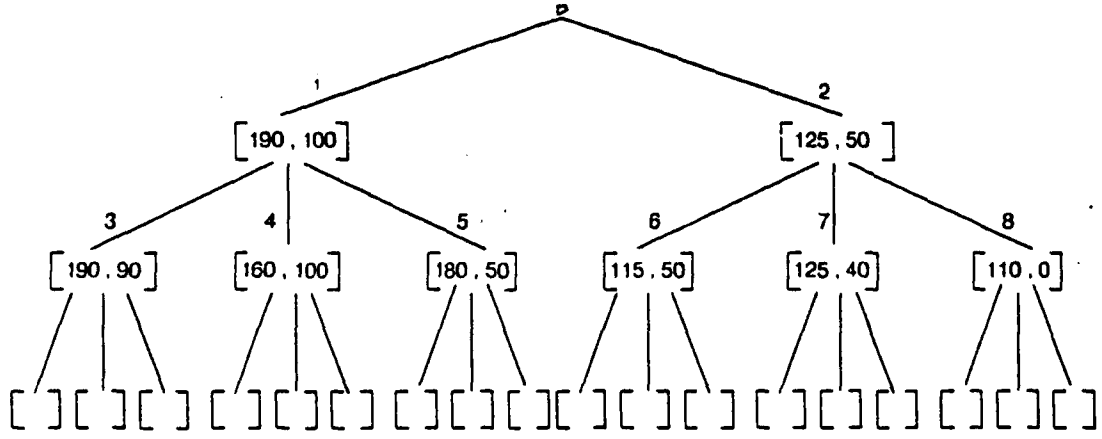


Figure 6-2: Non-Adversary Search Tree

Given the adversary tree in figure 6-1, in order to show that moving to node 1 is correct, one must either raise the pessimistic value of node 1 to at least 125 or one must lower the optimistic value of node 2 to no greater than 100. In order to raise node 1's pessimistic value, the optimistic values of nodes 3, 4 and 5 must all be lowered to -125. To do that one would have to show that the pessimistic value of one subnode of each of the nodes 3, 4 and 5 can be raised to at least 125. Thus in order to raise the pessimistic value of node 1, in the worst case, one would need to explore only three nodes at the third level of the tree.

In trying to lower the optimistic value of node 2 to no greater than 100, one would need to raise the pessimistic value of one of the nodes 6, 7 or 8 to at least -100. In the worst case, in order to accomplish that the optimistic values of all the subnodes of that node must be lowered to 100 or below. Thus, again, at the third level of the tree 3 nodes would have to be explored. The notion that the work involved in either strategy is about the same is a key to the success of the use of both strategies in B*.

In the non-adversary case the strategies in general will not have equal power. Figure 6-2 shows a non-adversary tree. In this case, in order to show that node 1 is the correct move, one must raise the pessimistic value of that node to at least 125. To do that one needs to be able to raise the pessimistic value of only one of the nodes 3, 4 or 5 to at least 125. In order to do that one needs to raise the pessimistic value of only one node on the third level of the tree.

However, in trying to reduce the optimistic value of node 2 to below 100, a different story unfolds. In order to reduce the optimistic value of node 2, one must reduce the optimistic values of all its subnodes. Then to reduce the optimistic values of those nodes, the optimistic value of all of their subnodes must be reduced. Thus at the third level of the tree presented in figure 6-2, 9 nodes would need to be explored. In general, the DISPROVEREST strategy leads to an exponential amount of work in a non-adversary search, whereas using the PROVEBEST strategy one needs only to find a single good path.

Independent of the cost of the DISPROVEREST strategy, it is likely that the strategy would not even be productive. Given a range $[\alpha,\beta]$ and n descendants within that range, it is likely that at least one of the descendants will have an optimistic value close to α. Thus the probability of reducing an optimistic value in a non-adversary search is not very high. On the other hand, it is highly unlikely that all the pessimistic values of the descendants will have a pessimistic value close to β. At least one of the descendants will have a value significantly greater than β. Thus while it is difficult to lower the optimistic value of a node, it is much easier to raise the pessimistic value of that same node.

That problem does not occur in adversary searches. In an adversary search, exploring one level of the tree affects the optimistic value of the parent, whereas at the the next level of the tree the pessimistic value of the root will be altered. If in figure 6-1, the original value of node 1 was [300,100], expanding node 1 would reduce the range to [200,100]. Then expanding any of the subnodes of node 1 would tend to raise the pessimistic value of node 1.

Although the full power of B* can not be enjoyed in a non-adversary search, it should be noted that the use of ranges and the ideas presented in section 4 on selection of nodes to be expanded at lower levels are still useful ideas.

7. Other Information to Guide the Search

Up to this point we have made our decisions using only the optimistic and pessimistic values of a node. No attempt has been made to use information that can be acquired from already expanded subtrees of a node under consideration.

Figure 7-1 presents an example of the inability of the range to properly reflect the true state of a node. In this case a choice by the player needs to be made. Both nodes have ranges of [200,0]. However the first node has three descendants with ranges of [0,-200], while the second node has only one relevant descendant also with a range of [0,-200]. Just using the range of a node to make a decision, these two nodes would be considered equal. Taking account of the subtrees of the two nodes would yield the correct selection of node 2.[4]

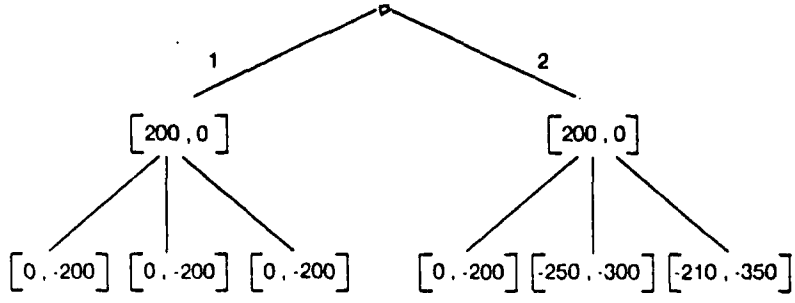


Figure 7-1:

Another example of the need to use information from subtrees is given in figure 7-2. Again both nodes have identical ranges of [300,100]. However, a look at the direct descendants of the nodes gives a different impression. Assume the goal of the search is to raise the pessimistic value of one of the nodes to above 250. In each case two nodes need to be explored. However, the probability of moving the optimistic value of both subnodes of node 1 to below -250 is greater than the probability of moving the optimistic values of the subnodes of node 2 to below -250. Thus the correct choice would be the first node.

Information can also be gathered from deeper locations in the subtrees. Figure 7-3 presents two nodes with identical sets of direct descendants. If one only examines the ranges of the direct descendants of the nodes in question (the subnodes of nodes 1 and 2), there is now difference between the two nodes. Both nodes have only one relevant subnode, each with a range of [0,-200]. However in examining the entire subtree of the first node, that subtree reduces to the subtree of node 1 in figure 7-1.

[4]Given that the goal is to raise the pessimistic value of one of the nodes above 0, it is much more likely that that one can lower the optimistic value of one node, instead of 3 nodes

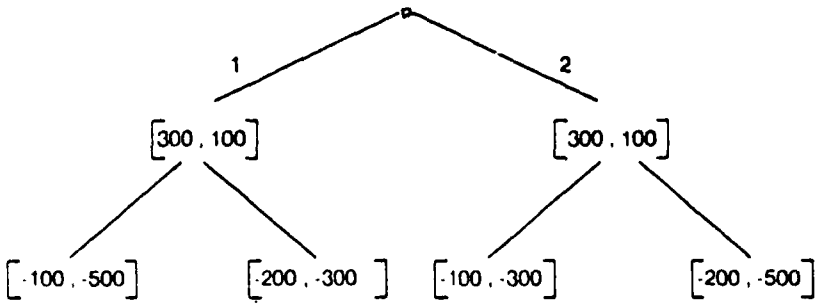


Figure 7-2:

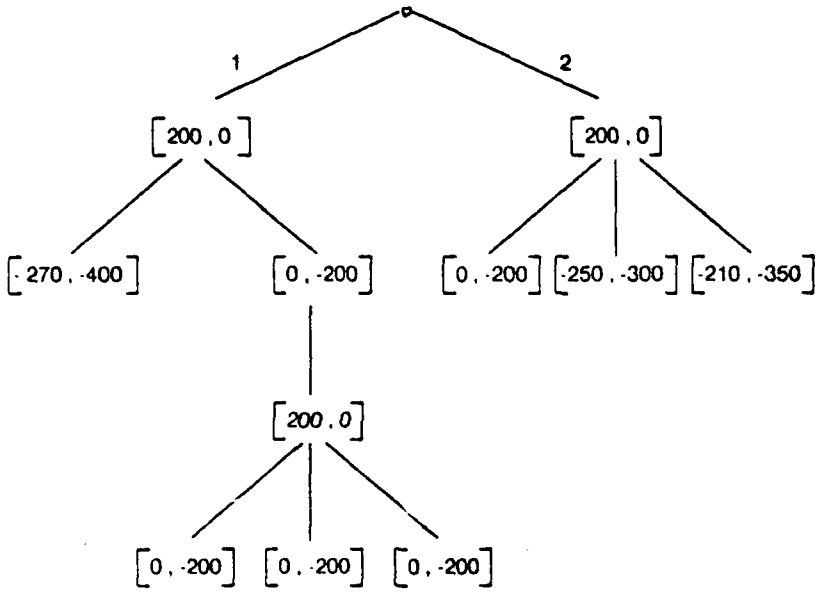


Figure 7-3:

At the present time it is not clear what information about subtrees is necessary to make the proper choice. One possible approach would be to try to approximate the expected true value of a node. For example, in figure 7-1, assuming that the true value of an unexpanded node could be any value within the range with equal probability, the expected true value of node 1 would be 50, where as the expected true value of node 2 would be 100. Thus using only that value, node 2 would be considered the best node.

Using only the expected true value can also be insufficient. If in figure 7-1 the nodes 1 and 2 had expected true values of 100 and 125 respectively, and the goal is to raise the pessimistic value of one of the nodes to at least 150, the expected true value measure would select node 2. Suppose that it has been determined that 95% of the time the true value of the first node would be found in the range of 175 and 25 while 95% of the time the true value of node 2 would be found in the range of 130 and 120. Since it is highly unlikely that pessimistic value of node 1 could be raised to 150 (at most 5% of

the time will the true value of node 2 be greater than 130), the best choice of nodes to explore would be node 1.

Another measure that has, for the most part, been ignored is the amount of work that needs to be done to accomplish a goal. In the original B* paper, Berliner used the depth of subtrees at the top level, in an attempt to gauge the amount of work that needed to be done. The measure that was used for the best node was the depth that the node's pessimistic value came from, while the measure used for the other nodes was the depth the respective optimistic values came from.

There are two problems with trying to approximate the amount of work to be done by a measure of depth. First, it highly discriminates against long but thin subtrees. Figure 7-4 presents a tree that effectively reduces to the tree in figure 7-5[5]. However, it is doubtful that any selection procedure that attempts to minimize depth would select node 2 even though very little work is truly involved. It could be argued that the case presented in figure 7-4 is highly unlikely and therefore depth still can be a reasonable measure. However, it is often the case that later in the search the above problem arises to some degree, and the use of depth often leads to the wrong choice.

The other problem with depth is that it does not provide a reasonable measure of work in non-uniform trees. It is not realistic to assume that the branching factor at every node is the same. If one subtree has an average branching factor of 20 and another subtree has an average branching factor of 3, a depth measure will not reflect the difference.

A better measure would be the number of terminal nodes that need to be expanded in order to change one of the bounds of the current node. The exact number of expansions can not be calculated a priori, since expanding one node may lead to further expansions of its descendants. The minimum number of expansions can be calculated.

In fact, one would really need the minimum number of nodes that would have to be expanded in order to change one of its bounds to a given value. In figure 7-6 the information that a minimum of 1 node needs to be expanded to raise the pessimistic value of the root above 0, is of little value if the goal is to raise the pessimistic value of the root to above 50. In that case, it would be necessary to know that a minimum of 3 expansions are needed. At every node, a list containing the minimum number of expansions needed in order to reach a given value would be kept. The list associated with the root of figure 7-6 is given in figure 7-7. Unfortunately, the list at a node grows in direct relation to the number of leaf nodes in the subtree. Thus the cost of storage would be prohibitive.

---

[5]This is based on the assumption that the cost of moving between already expanded nodes is overshadowed by the cost of expanding nodes and determining the ranges of the descendants

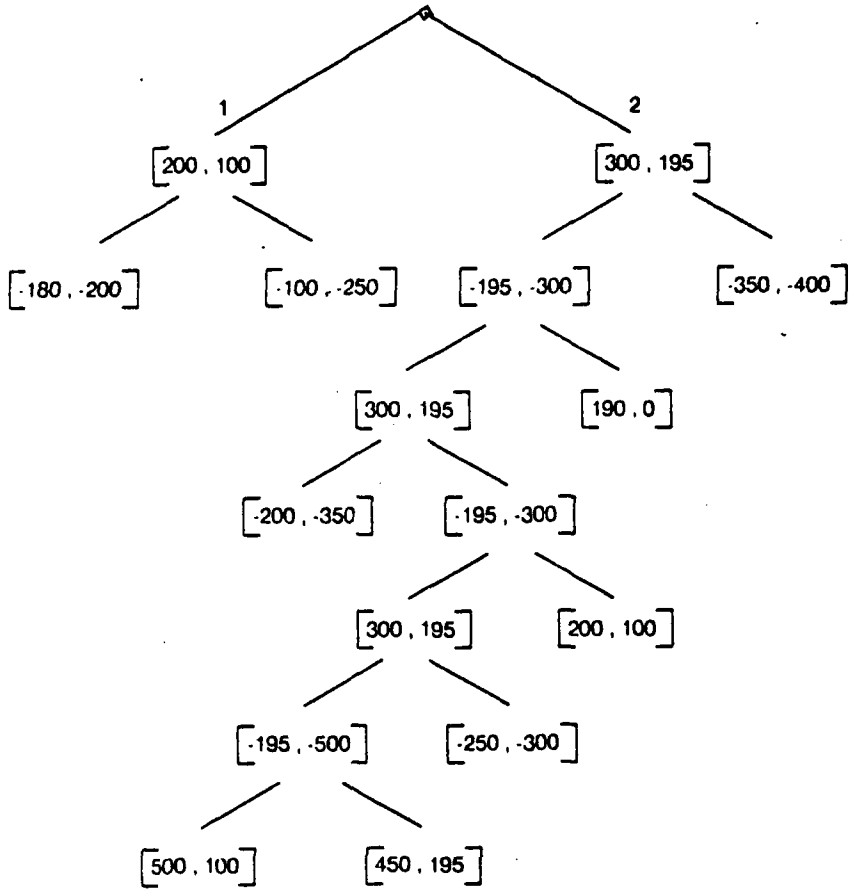


Figure 7-4: A Thin Tree

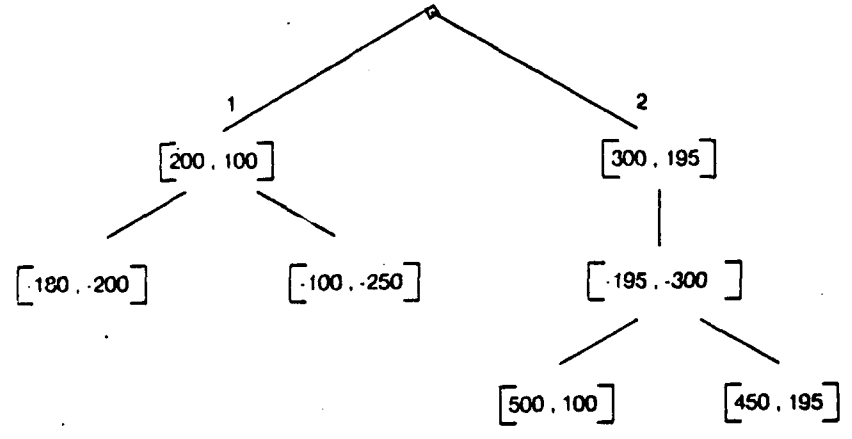


Figure 7-5: A Thin Tree - Reduced

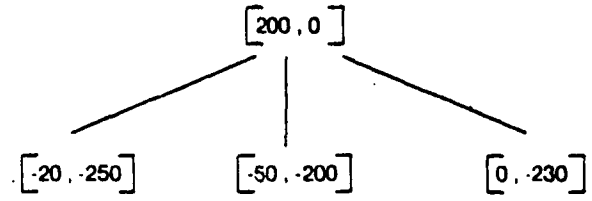


Figure 7-6:

Optimistic Value	Pessimistic Value
Minimum of 1 to move below 200	Minimum of 1 to move above 0
	Minimum of 2 to move above 20
	Minimum of 3 to move above 50

Figure 7-7: A Work List for the Root of Tree in Figure 7-6

Finally, even if one develops a reasonable measure for work, it is not clear that using the measure would be beneficial. It can be used in the situation when the probability of reaching the goal is the same for two or more nodes. In this case one would like to be able to choose the node that will require the least amount of work. Those cases do not arise very often, so unless the computation of the work measure is inexpensive in both time and space, the use of a work measure can hardly be justified.

Another possible use of the notion of work is in guiding the search to less probable nodes that require very little work in order to reach a goal (if in fact it can). If the most probable node requires a large expenditure of work then one might want to examine less probable nodes that require small amounts of work, since that extra work will only increase the total amount of work that truely needs to be done by a small factor. Where that type of selection is reasonable is an open question. However, it would seem to require a good notion of where the true value of a node is to be found before a good determination based on work can be made.

8. When is B* Useful?

There are certain types of problems where the B* paradigm is not effective. In particular, B* does not seem to be suited for finding solutions to puzzles. However, B* does appear to be well suited for chess. Protocols of chess masters, collected by DeGroot [DeGroot 65] appear to support the B* paradigm.

Berliner attempted to use the B* algorithm to find solutions to the 8-puzzle [Nilsson 71]. Compared to a best-first search, the B* search provided no gain. A difference between the 8-puzzle and chess problems is that the goal of the 8-puzzle is to find a complete line of play while in chess only the next move is desired.

In developing a solution to the 8-puzzle, a B* search will eventually have to explore the node with the highest optimistic value at each level of the tree[6]. Postponing the exploration of those nodes, as is done by the lower level selection procedure, will only result in the useless exploration of the other nodes.

The B* paradigm is useful whenever the position in which the player finds himself the next time he must make a move is not solely determined by the current move he makes. In an adversary search, the opponent makes a move after the player, so it is impossible to guarantee the next position the player will find himself. In most puzzles, that is not the case.

[6]The node with the highest optimistic value need not be explored in determining the best move at the root; however, in determining the total line of play the best node at each level will eventually be explored by some application of the B* algorithm.

9. Bounding Functions and the Horizon Effect

The major open issue about the B* algorithm is the development of bounding functions. In the course of this research, we have not attempted to devise any bounding functions, but we have examined analytically the effect on the B* paradigm when the bounds of a new node are not within the bounds of its parent. In particular, if the bounds of a node, do not strictly bound the true value, the horizon effect [Berliner 73] can result.

The horizon effect is a condition that occurs when a search procedure pushes, by selecting delaying moves, the eventual outcome of a move at the root of the search tree far enough away from the root that it disappears from the scope of the search. Thus, the value reported for that move no longer reflects the eventual outcome of making that move.

If the bounds of a node are strictly bounding, the node returned by the algorithm is guaranteed to be the best. The pessimistic value of the node will be the absolute lowest value the true value of the node can be. However, if the bounds are not strictly bounding then it is possible to delay finding the true low value of the returned best node past the horizon of the search.

Figure 9-1 presents a non-adversary search without values being backed-up where the bounding function is not strictly bounding. The goal at the root is to raise the pessimistic value of node 2 above 150. If the player selects node 2, eventually, he must reach a node with a maximum value of 50. However, by the time the search expands node 6, the backed up pessimistic value of node 2 will be 150. The search will terminate and return node 2 as the best node, even the though true value of node 2 is at best 50. In this way the search procedure has pushed the bad effect of the move to node 2 past the horizon of the search.

It is doubtful that a bounding function will be strictly bounding. More likely, the bounds of a node will be invalid a small percentage of the time. It is not clear how small that percentage must be. Even with an extremely small percentage of invalid nodes, it might be the case that a large percentage of those cases result in the horizon effect. Only by examining the use of the B* algorithm on a real problem, can the effect of invalid bounds be measured.

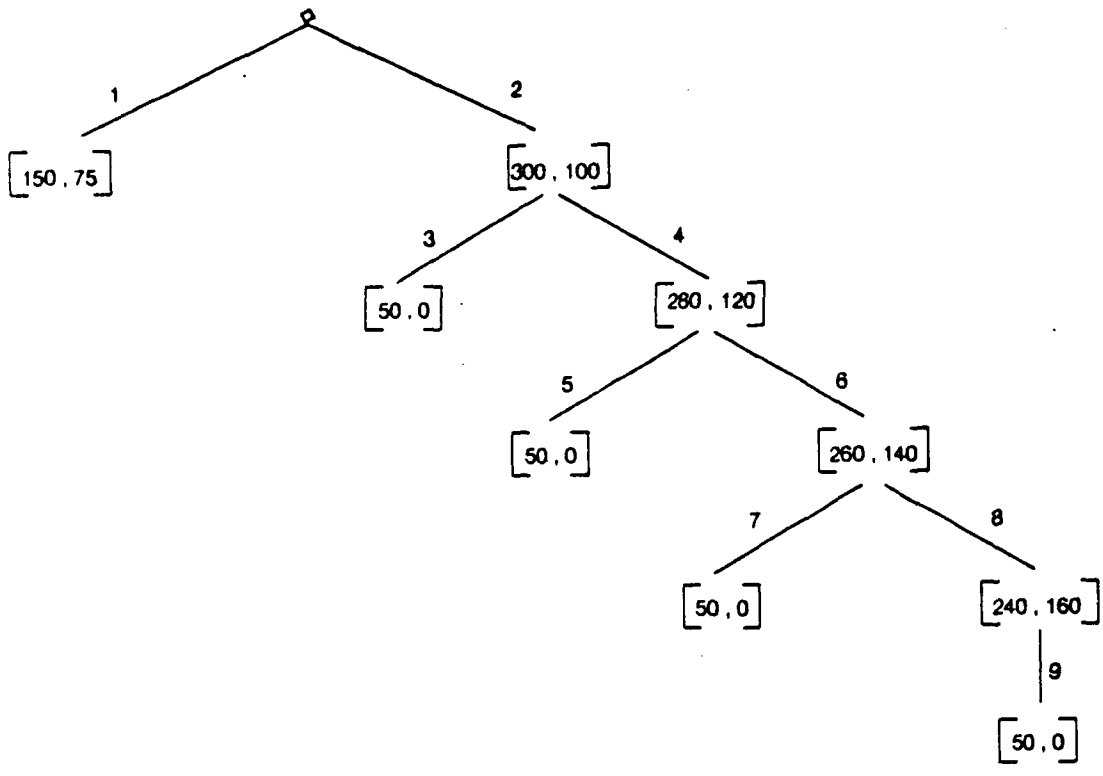

1
2
[150 , 75]
[300 , 100]
3
4
[50 , 0]
[280 , 120]
5
6
[50 , 0]
[260 , 140]
7
8
[50 , 0]
[240 , 160]
9
[50 , 0]

Figure 9-1:

10. Conclusion

We have presented a set of rules for making decisions throughout the B* search algorithm. These rules provide a savings of 65% of the work required by a best-first search. The rules also require 35% less work than the best version of the B* algorithm presented so far. We have also explored the selection procedure at the lower levels of the search and showed the rules developed save 25 % of the number of nodes expanded in a procedure that always selects the best node and save 35 to 40% in the number of nodes explored.

The rules are based on a simple probability model. It is hoped that by extending the model to allow for the backing up of values from lower levels of the tree (as described in section 7), additional speed up will be provided. The extension of the probability model may also provide an insight to the way humans search. It seems clear that humans maintain additional information about a node in a search other than one or two values. We conjecture that if the B* algorithm does correctly capture the technique of human searching, then a probability model will provide the additional information used by humans in guiding the search.

Acknowledgement

The helpful comments of Hans Berliner, Elaine Kant and Bruce Ladendorf are gratefully acknowledged.

Bibliography

[Berliner 73] Berliner, H.J.
Some Necessary Conditions for a Master Chess Program.
In *Third International Joint Conference on Artificial Intelligence.* , Stanford University, 1973.

[Berliner 79] Berliner, H.J.
The B* Tree Search Algorithm: A Best First Proof Procedure.
Artificial Intelligence 12, 1979.

[DeGroot 65] DeGroot, A.D.
Thought and Choice in Chess.
Mouton and Co., Der Haag, 1965.

[Nilsson 71] Nilsson, N.J.
Problem-Solving Methods in Artificial Intelligence.
McGraw-Hill, New York, 1971.

[Palay 80] Palay, A.J.
A Procedure for Calculating Optimum B* Search Trees.
1980.
Available from Author.

I. Calculation of Optimum Trees

Given a B* search tree, an optimal tree is defined as the minimal set of nodes that need to be expanded in order to prove that the best node is, indeed, the best node. It should be noted that the optimum tree for a given search tree is not necessarily unique.

The procedure for determining the size of the optimum B* trees consists of two parts. The first part determines which node is the best node. Given the best node, the second part determines the size of the optimum tree. Since the best node for each simulated tree was already determined in earlier runs of the B* algorithm the first part needed only to read in the computed best node from a file. If those values were not known, a priori, then a version of the B* algorithm would have to be used to determine the best node.

In determining the size of the optimum tree, the second part of the procedure determines the optimum separation point for the proof. At the root of the tree, two lists are maintained. One list (Best-Node Work List) describes the minimum number of node expansions that need to be done to raise the pessimistic value of the best value to a given point. The other list (Alternative Work List) describes the minimum number of node expansions that need to be expanded in order to lower the pessimistic values of the alternative nodes below a given point. Given the two lists, the size of the optimum tree can be determined by finding the separation point that minimizes the sum of the number of expansions that are required to raise the pessimistic value of the best node above that point and the number of expansions required to lower the optimistic values of the alternative nodes below that separation point. The procedure that determines the size of the optimum tree, given the two lists, is presented in figure I-1.

In order to create those two lists, two additional values are maintained for each node in the search tree. The two values will be referred to as WORK-DONE and WORK-NEEDED. For the best node, the WORK-DONE value will contain the number of nodes that have been expanded in order to raise the pessimistic value of the node to its current value. The WORK-NEEDED value contains the minimal number of nodes (including the ones already expanded) that will have to be expanded in order to raise the pessimistic value further. For nodes at the player's level in the best node's subtree, the values are the same as above. For nodes at the opponent's level of the best node's sub-tree, the WORK-DONE value contains the number of nodes that have been expanded in order to lower the optimistic value of the node to its current value. The WORK-NEEDED value contains the minimal number of expansions needed in order to lower the optimistic value further.

For the alternative nodes, the WORK-DONE value contains the number of nodes that have been expanded in order to lower the optimistic value of the node to its current value. The WORK-NEEDED value contains the minimal number of expansions needed in order to lower the optimistic value

```
PROCEDURE FIND!SIZE!OF!TREE(best!node!work!list,alternative!work!list);
BEGIN

  size!of!optimum!tree ← INFINITY;

  alt!pointer ← size of alternative!work!list;
  best!pointer ← start of best!node!work!list;

  WHILE best!pointer ≤ size of best!node!work!list AND
        alt!pointer ≥ start of alternative!work!list DO
    BEGIN

      WHILE  best!pointer ≤ size of best!node!work!list AND
        new!pessimistic!value[best!pointer] < new!max!optimistic[alt!pointer] DO
          best!pointer ← best!pointer + 1;

      WHILE alt!pointer > start of alternative!work!list AND
        new!max!optimistic[alt!pointer-1] < new!pessimistic!value[best!pointer] DO
          alt!pointer←alt!pointer-1;

      new!size ← # of expansions of best!node!work!list[best!pointer] +
                 #.of expansions of alternative!work!list[alt!pointer];

      size!of!optimum!tree ← MIN(new!size,size!of!optimum!tree);

      alt!pointer←alt!pointer-1;

    END;

  RETURN(size!of!optimum!tree);

END;
```

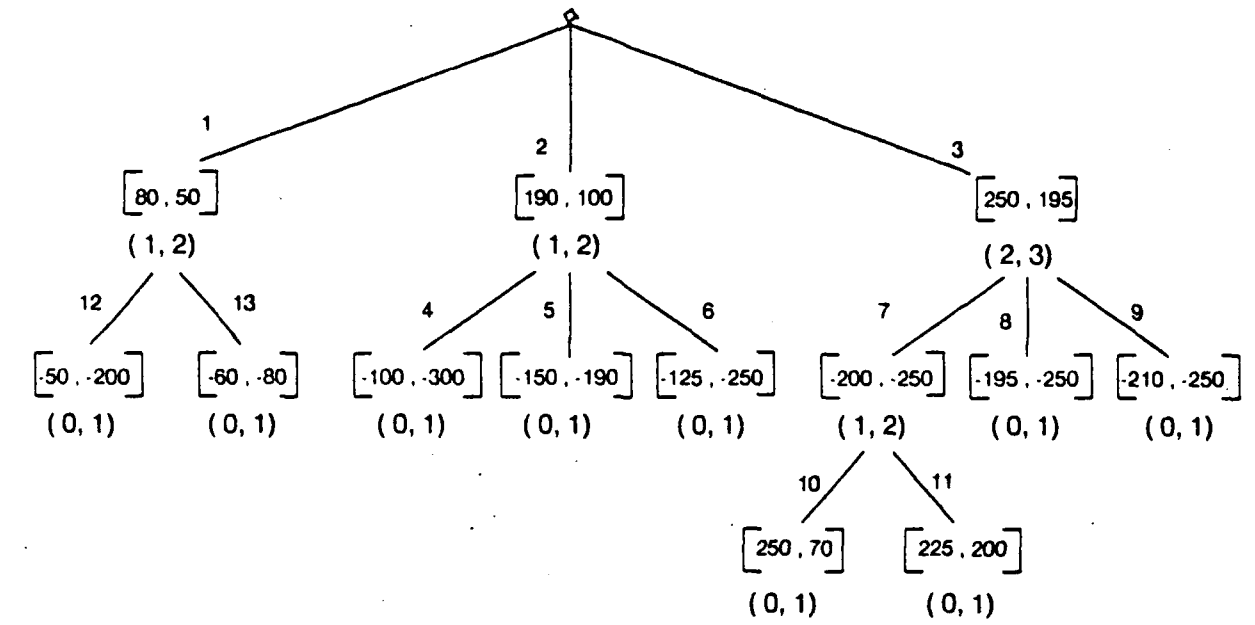
Figure I-1: Procedure for Calculating Size of Optimum Tree

further. For nodes at the player's level in an alternative node's subtree the values refer to the lowering of the optimistic value. At the opponent's level the values refer to the raising of the pessimistic value.

In order to form the list for the alternative nodes at the top level of the tree, two additional values are maintained. The value MAX-OPTIMISTIC contains the maximum optimistic value of the alternative nodes. The value, MAX-NEED, contains the minimum number of nodes that need to be expanded in the alternative nodes' subtrees in order to reduce the value of MAX-OPTIMISTIC.

In order to determine the size of the optimum tree, the procedure alternately searches the best node's subtree and the subtrees of the alternative nodes. Whenever the pessimistic value of the best node is raised, an entry in the work list for the best node is made. Whenever MAX-OPTIMISTIC is lowered an entry for the work list of the alternative nodes is made.

Figure I-2 presents the status of the optimum search procedure on the tree presented in section 2. Node 3 is the best node. The optimistic and pessimistic values are presented (enclosed in brackets) for each node along with the WORK-DONE and WORK-NEEDED values (enclosed in parentheses). The original pessimistic value of node 3 was 0. In order to raise the pessimistic value of node 3 to 195, two nodes had to be expanded (nodes 3 and 7). In order to raise the pessimistic value of node 3 further, at



Best-Node Work List

# of expansions	new pessimistic value
START	0
2	195

Alternative Work List

# of expansions	new MAX-OPTIMISTIC
START	300
1	200
2	190

Figure I-2: Calculation of an Optimum B* Tree

least one additional node (node 8) will have to be expanded. For the alternative nodes, the original value of MAX-OPTIMISTIC was 300. By expanding node 2 the value of MAX-OPTIMISTIC was lowered to 200. Then by expanding node 1, MAX-OPTIMISTIC was lowered to 190. In order to lower MAX-OPTIMISTIC further, at least one more node will have to be expanded (either node 4, 5, or 6). Thus the current value of MAX-NEED is 3.

When the procedure finds a separation point, a maximum size for the optimum tree is determined by the sum of the WORK-DONE values of all the top level nodes plus one for the expansion of the root. In the example, the maximum size of the optimum tree is 5 expansions. Given the maximum size for the optimum tree, the procedure searches the best node's subtree until the WORK-NEEDED value of the best node is greater than or equal to that maximum size. The procedure then searches the alternative nodes until MAX-NEED is greater than or equal to that maximum size.

By scanning the two lists, the optimum separation point can be determined, thus yielding the size of the optimum tree. For a complete description of the procedure for generating optimum B* trees, see [Palay 80].